APPENDIX

CONVERTER CONFIGURATION AND TRANSMISSION FORMAT

Introduction

This document describes the data structures and packet flow between the system (e.g., a PC) and the converter / FPGA, the so-called “discrete” (i.e. non-ASIC) implementation of the converter.

The FPGA is an integrated circuit holding custom logic. It adds some infrastructure (memory, mostly) for a PCI-based dataflow between the OHCI controller and a USB2 I/F chip inside the converter device.

Due to restrictions of the USB I/F, the discrete implementation of the converter operates in different modes to deliver IEEE1394 and AV functionality over USB. Both modes are mutually exclusive, i.e. only one set of functionality is active at any given time.

Note: **Physical transmission order** on USB is “little-endian”, i.e. “least significant byte first”. The notation in this document is “most significant byte first”, and might lead to assuming a different transmission order if read from left to right. That’s not the case, however - it **is right to left.**

USB Configuration

The converter exhibits the following configuration over the USB bus:

Full speed mode:

Interface Descriptor 0 – alternate setting 0

Endpoint	Direction	Type	Byte / Packet	Purpose
0	*IN/OUT*	*CONTROL*	*64*	*Firmware download*
1	OUT	BULK	64	FX2 – cmd (i2c, etc.)
1	IN	BULK	64	FX2 – response (i2c, etc.)

High speed mode:

Interface 0 – alternate setting 0 – Idle

Endpoint	Direction	Type	Byte / Packet	Purpose
0	*IN/OUT*	*CONTROL*	*64*	*Firmware / FPGA Download, i2c, mode selection*
1	OUT	BULK	512 (64)	FX2 – cmd (i2c, etc.)
1	IN	BULK	512 (64)	FX2 – response (i2c, etc.)
2	OUT	BULK	512	FPGA Download

Interface 0 – alternate setting 1 – 1394

Endpoint	Direction	Type	Byte / Packet	Purpose
0	*IN/OUT*	*CONTROL*	*64*	*Firmware / FPGA Download, i2c, mode selection*
1	OUT	BULK	512 (64)	FX2 – cmd (i2c, etc.)
1	IN	BULK	512 (64)	FX2 – response (i2c, etc.)
2	OUT	BULK	512	FPGA Packets (Control, Async)
4	IN	BULK	512	FPGA Packets (Ack, Async)
6	OUT	BULK	512	FPGA Packets (Isoch.)
8	IN	BULK	512	FPGA Packets (Isoch.)

Interface 0 – alternate setting 2 – AV Playback

Endpoint	Direction	Type	Byte / Packet	Purpose
0	*IN/OUT*	*CONTROL*	*64*	*Firmware / FPGA Download, i2c, mode selection*
1	OUT	BULK	512 (64)	FX2 – cmd (i2c, etc.)
1	IN	BULK	512 (64)	FX2 – response (i2c, etc.)
2	OUT	BULK	512	Video data
6	OUT	BULK	512	Audio data

Interface 0 – alternate setting 3 – AV Capture

Endpoint	Direction	Type	Byte / Packet	Purpose
0	*IN/OUT*	*CONTROL*	*64*	*Firmware / FPGA Download, i2c, mode selection*
1	OUT	BULK	512 (64)	FX2 – cmd (i2c, etc.)
1	IN	BULK	512 (64)	FX2 – response (i2c, etc.)
2	IN	BULK	512	Video data
6	IN	BULK	512	Audio data

Note: The actual maximum packet size may differ from the value reported in the device descriptor. If applicable, the actual value is noted in brackets.

Global mechansims

All mode-independent communication between the system and device is transferred using endpoint 1 (implemented by two BULK pipes). Exceptions to this rule are standard USB requests (e.g. "select alternate setting") and FX2 firmware download, which use endpoint 0. Both endpoints (EP0 and EP1) are thus present in all alternate settings.

Endpoint 1 transactions

Endpoint 1 supports the transactions described below. Please note that endpoint 1 follows a "request/response" scheme, i.e. all actions by the system are matched with an acknowledge and/or data packet in response. These packets start with echoing the command ID.

Notation: The tables below describe the packet content for requests and responses from the external device. Offsets are counted in byte. Non-byte fields (words, dwords) are expected to be transferred in "little-endian" (LSB first) orientation.

i²c write

Offset	Field	Size	Value
			Request (system to device)
0	command	1	0x01
1	device address	1	(see below)
2	data length	1	
3	data	data length	
			Response (device to system)
0	command	1	0x01
1	success	1	(boolean)

Note: If the targeted device uses subaddresses, the additional addressing information needs to be put at the start of the data section.

i²c read

Offset	Field	Size	Value
			Request (system to device)
0	command	1	0x02
1	device address	1	
2	data length	1	
3	subaddr len	1	
4	subaddress	subaddr len	
			Response (device to system)
0	command	1	0x02
1	success	1	(boolean)
2	data	data length	

If the read transaction failed, no data will be returned (data field size is 0 bytes).

Note: If the targeted device does not use subaddresses, set "subaddr len" to 0.

Set Reset

Offset	Field	Size	Value
			Request (system to device)
0	command	1	0x03
1	device ID	1	
			Response (device to system)
0	command	1	0x03
1	reset state	1	

Device ID: (same as i2c address)

Clear Reset

Offset	Field	Size	Value
			Request (system to device)
0	Command	1	0x04
1	device ID	1	
			Response (device to system)
0	Command	1	0x04
1	reset state	1	

Device ID: (same as i2c address)

Start FPGA Download

Offset	Field	Size	Value
			Request (system to device)
0	command	1	0x05
			Response (device to system)
0	command	1	0x05
1	success	1	(boolean)

Note: This command is only available while alternate setting 0 (idle mode) is active.

Get FPGA State

Offset	Field	Size	Value
			Request (system to device)
0	command	1	0x06
			Response (device to system)
0	command	1	0x06
1	firmware present	1	(boolean)

Get USB Speed

Offset	Field	Size	Value
			Request (system to device)
0	command	1	0x07
			Response (device to system)
0	command	1	0x07
1	speed	1	(see below)

Speed:

- 0: Fullspeed (12 Mbit/s)
- 1: Highspeed (480 Mbit/s)

Flush Endpoint FIFOs

Offset	Field	Size	Value
			Request (system to device)
0	command	1	0x08
			Response (device to system)
0	command	1	0x08

Clears FIFOs for endpoints 2, 4, 6, 8 in Cypress FX2 chip.

Read Timestamp

Offset	Field	Size	Value
			Request (system to device)
0	command	1	0x09
			Response (device to system)
0	command	1	0x09
1	timestamp	8	64 bits from the FPGA

Response packet contains latest timestamp from the FPGA.

Note: This command is only valid in playback mode.

Echo

Offset	Field	Size	Value
			Request (system to device)
0	command	1	0xE0
1	token	1	
			Response (device to system)
0	command	1	0xE0
1	token	1	

This is a "NOP" command, which just get's echoed back to the system (including the token). It's purpose is to help flushing the USB I/F pipes after device discovery, eliminating any residue from previous sessions.
Suggested squence on driver load:

- Send "Echo" command to device, token = 1
- Read from EP1 until "Echo" with token = 1 is received
- Send "Echo" command to device, token = 2
- Read from EP1 until "Echo" with token = 2 is received

Mode switching

Selecting a converter operation mode (1394, AV Capture, AV Playback) requires three steps:

- Put device into Idle mode – select alternate setting 0.
- Download the FPGA configuration for the new operation mode.
- Put device into new operation mode – select appropriate alternate setting.

FX2 Firmware download

Firmware download to the FX2 microcontroller is implemented nativly in FX2 device, using endpoint 0; see separate documentation from Cypress for details.

FPGA download

The download of FPGA configuration data is only supported in "idle" mode (alternate setting 0). Then EP2 becomes available to carry the FPGA download data.

i²c addresses

The following table lists all i²c devices on the converter with their respective address:

Address	Device
0x4A	Video Decoder SAA7113
0x54	Video Encoder ADV7179
0xF0	FPGA
0xF2	*AC97 (set/clear reset, only)*
0xA0	EEprom 24C0x

EEprom content

Offset	Length	Content
0x00	1	0xC2
0x01	2	Vendor ID
0x03	2	Product ID
0x05	2	Device ID
0x07	1	Configuration Byte: 0x40
0x08	2	Code Length: 0x03
0x0A	2	Start Address: 0x00
0x0C	3	8051 Code
0x0F	2	Code Length: 0x47 (71)
0x11	2	Start Address: 0x80
0x13	71	8051 Code
0x5A	5	Lead Out
0x70	8	Serial number (from label)
0x78	8	GUID for IEEE1394

1394 mode

This section of the document does only describe the *mechanisms* for data exchange between the OHCI controller and the system in the discrete converter implementation. It does not describe the actual content and structure of this communication flow. For further details please refer to the "1394 Open Host Controller Interface Specification" (Release 1.1, January 6, 2000).

IEEE1394 connectivity for the system is achieved by "remote controlling" the PCI-based OHCI controller in the converter. The FPGA offers the following resources and mechanisms to help mimic a PC-like PCI environment inside the converter:

- 4 Kbyte of RAM: OHCI controller can read from this memory in true random fashion. The system can write content (OHCI descriptors, IEEE1394 packets) into the memory over USB. Any data written to this area from OHCI controller gets forwarded to the system, as well as updated in FPGA RAM.
- FIFO-mapped (pseudo) RAM: 2 Kbyte of FIFO are mapped to a memory range on PCI. All writes to these locations are forwarded to the system.
- PCI master functionality: The system can initiate (over USB) read and write transactions on the local PCI bus inside the converter. Each access is 32 bits wide. This allows management of all OHCI registers.

All events requiring realtime responses are handled by the microcontroller in the USB I/F. This includes any interrupts (if necessary), and some aspects of isochronous packet streaming.

The packets are assumed to be transmitted using BULK transactions on USB. This assures data integrity and hardware-based flow control.

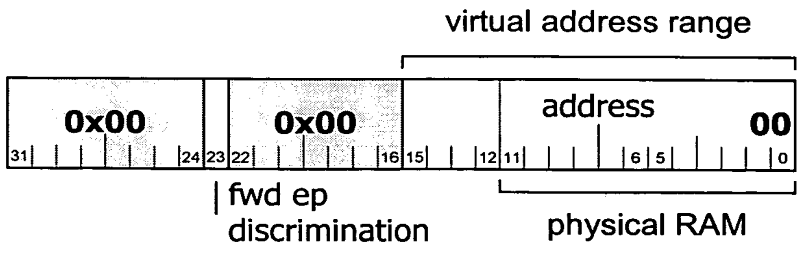
The USB controller will forward any incoming USB packets to the FPGA, either unconditionally (asynchronous packets) or time-based (isochronous packets).

The FPGA handles the packetization of PCI DMA data when sent to the system over USB. This includes insertion of a leading header, carrying the original PCI address and the length of the following burst. The FPGA employs some simple "burst-combining" where possible.

PCI Addresses

The embedded PCI bus between the FPGA and the OHCI controller supports the following address spaces:

<u>FPGA Target access</u>



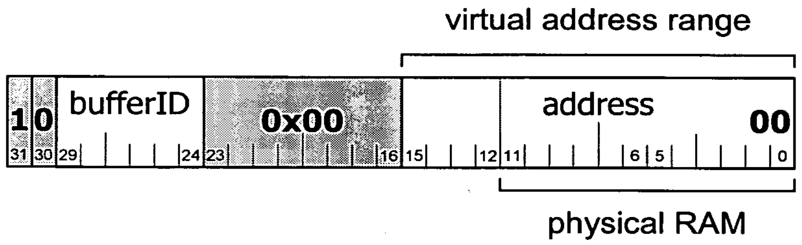
To address the physical RAM inside the FPGA, the upper half (esp. bit 17) shall be left clear. The lower 12 bit select the actual phyiscal RAM location. DMA forwarding packets (resulting from writes to the FPGA address space) echo the complete lower half (16 bits) of the address.

Read transactions: The physical RAM is "shadowed" throughout the complete virtual address range (bits 15 to 12 are disregarded for address decoding).

Write transactions: All write accesses are forwarded to the system. The physical RAM is only updated in the range of 0x1000 to 0x1FFF. Depending on address bit 23, the forwarding endpoint is chosen as follows:

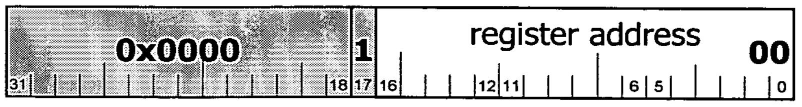
Bit 23	Endpoint	transaction size	commit timeout
0	EP4	512 bytes	7 µs
1	EP8 (Isoch. Data)	4096 bytes	300 µs

FPGA Target access with buffer ID



To support flow control for outbound data streams, the FPGA supports a "buffer" mechanism. The buffer ID is encoded in the upper byte of the PCI address (see diagram above). It is independent of the actual phyiscal or "virtual" address in the lower 16 bits.
The FPGA detects a buffer access when bit 31 of the address is set. This applies to read and write transactions.

OHCI register access



Whenever bit 17 is set, the PCI transaction targets the OHCI controller (i.e. its registers).

Packet format

All packets to and from the FPGA have a four byte header. The four uppermost bits of this dword identify the packet type.

Note that the FPGA does not see USB packet boundaries. It just receives a stream of logical packets as defined below. Thus, multiple "FPGA packets" can be concatenated and transmitted together in a single USB packet. Spreading a single

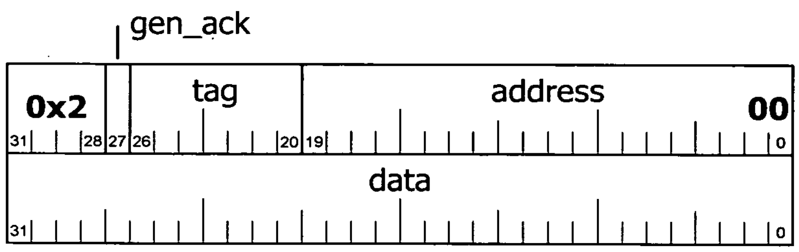
"FPGA packet" over multiple USB packets is theoretically possible, but strongly discouraged.

Packet Code Reference

Code	Type
0x2	PCI WRITE DWORD
0x3	PCI READ DWORD
0x4	FPGA REGISTER WRITE
0x5	FPGA REGISTER READ
0x6	PCI CONFIG WRITE DWORD
0x7	PCI CONFIG READ DWORD
0x8	STORE IN RAM
0x9	PCI DMA FORWARDING
0xA	PCI INTERRUPT FORWARDING
0xB	WAIT FOR BUFFER ACCESS

PCI WRITE DWORD (0x2)

Request (from System)



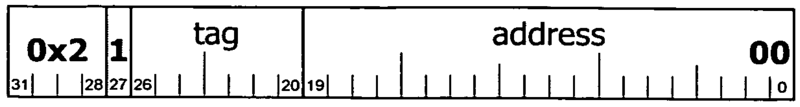
This packet instructs the FPGA to initiate a master write command cycle on the local PCI bus.

The dword *data* gets written to *address*. *Address* is dword-aligned.

If *gen_ack* is set to 0, *tag* is ignored and no further actions are taken.
If *gen_ack* is set to 1, the FPGA will generate a response packet (see below).

Note: If this packet was sent on the isoch transmit endpoint, the PCI transaction is only performed when the *isoch_store_disable* flag is cleared (see the FPGA Register). The acknowledge packet (if requested) is generated regardless of the flag status.

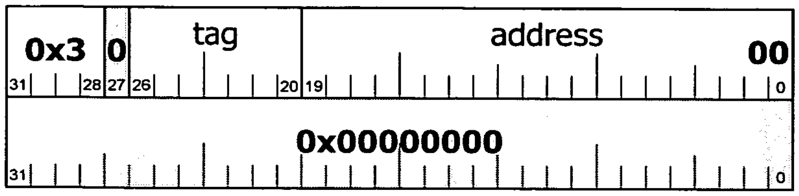
Response (from the converter)



The *tag* and *address* fields repeat the values from the original request packet.

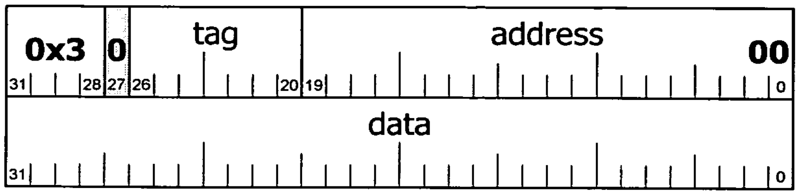
PCI READ DWORD (0x3)

Request (from System)



This packet instructs the FPGA to initiate a master read command cycle on the local PCI bus. A dword of data gets read from *address* (dword-aligned). The FPGA will generate a response packet with the data read from the target (see below).

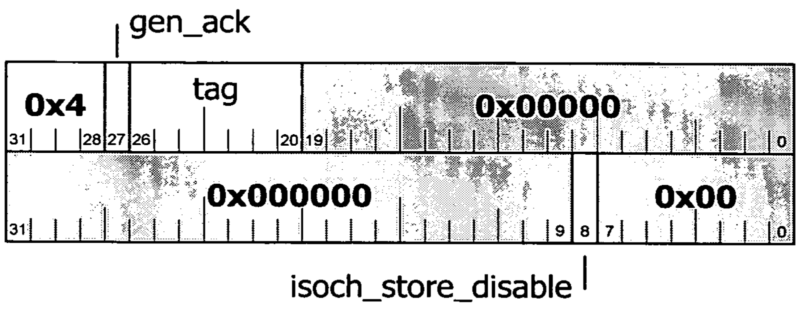
Response (from the converter)



The *tag* and *address* fields echo the values from the original request packet. *Data* contains the value read from the PCI target.

FPGA REGISTER WRITE DWORD (0x4)

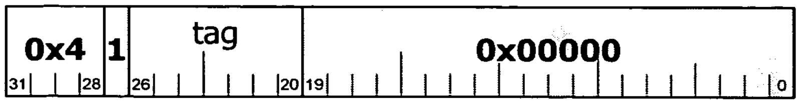
Request (from System)



This packet type allows access to an internal register inside the FPGA.

If *isoch_store_disable* is set, all STORE IN RAM, PCI WRITE DWORD and WAIT FOR BUFFER ACCESS packets on the isochronous OUT endpoint are skipped; in addition, the "wait for buffer" mechansim is disabled.
If *isoch_store_disable* is clear, normal processing of packets on the isochronous OUT endpoint is resumed.

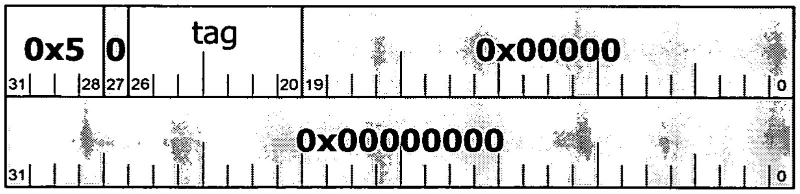
Response (from the converter)



If requested, the FPGA will generate this acknowledge packet for a register write command.

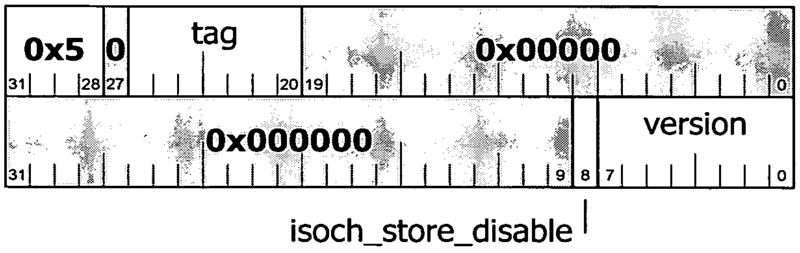
FPGA REGISTER READ DWORD (0x5)

Request (from System)



This packet requests the current status of the internal FPGA register (see below).

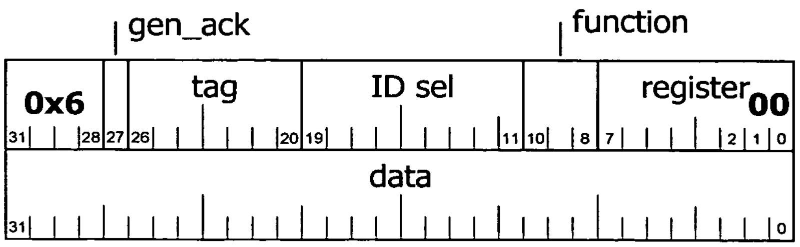
Response (from the converter)



This packet contains the current status of *isoch_store_disable* as well as the current version number of the FPGA firmware.

PCI WRITE CONFIG DWORD (0x6)

Request (from System)

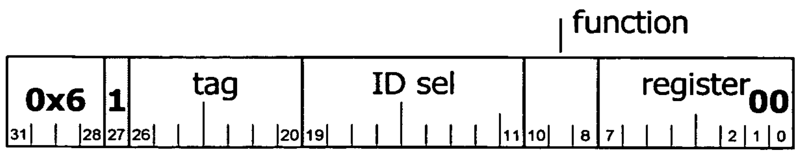


Similar to PCI WRITE DWORD, but instead of a MEM WRITE, a CONFIG WRITE transaction is generated on PCI.

The dword *data* gets written to the specific *register* and *function* of the PCI device selected by *ID sel.* Only one bit may be set in *ID sel.*

If *gen_ack* is set to 0, *tag* is ignored and no further actions are taken.
If *gen_ack* is set to 1, the FPGA will generate a response packet (see below).

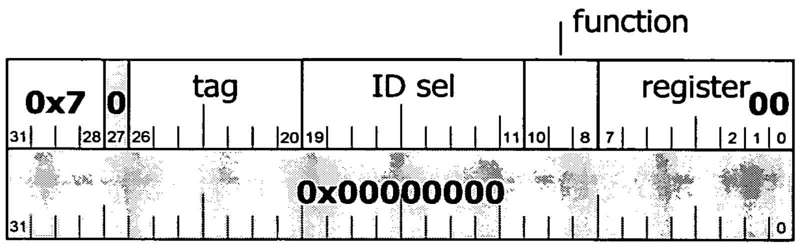
Response (from the converter)



The *tag, ID sel, function* and *register* fields repeat the values from the original request packet.

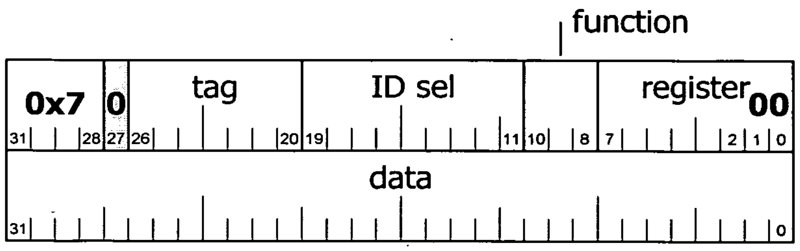
PCI CONFIG READ DWORD (0x7)

Request (from System)



This packet instructs the FPGA to initiate a CONFIG READ cycle on the local PCI bus. A dword of data gets read from the specified *register* and *function* of the PCI device selected by *ID sel*. The FPGA will generate a response packet with the data it read from the target (see below).

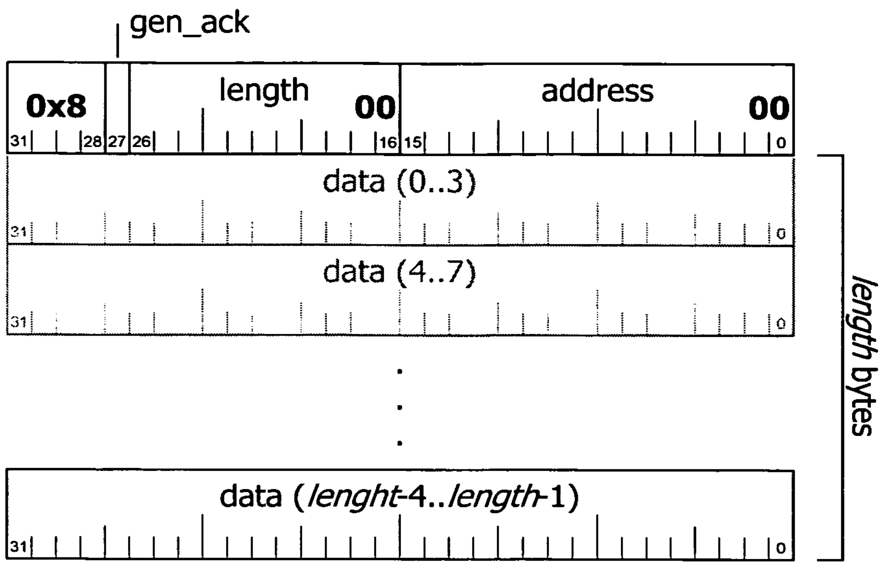
Response (from the converter)



The *tag*, *ID sel*, *function* and *register* fields repeat the values from the original request packet. data contains the value read from the PCI target

STORE IN RAM (0x8)

Request (from System)



This packet holds data that will be written to FPGA RAM, and are subsequently available for read operations originating from the OHCI controller.

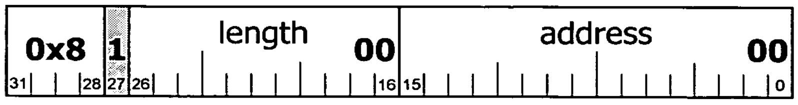
The *data* following the header gets written to FPGA RAM, starting at *address* (dword aligned). There are *length* bytes of data following the header; *length* is a multiple of 4.

Note: *address* is only valid in the range as mapped on PCI, as defined above.

Note: If this packet was sent on the isoch_transmit endpoint, the update of the RAM content depends on the *isoch_store_disable* flag (see the FPGA Register).

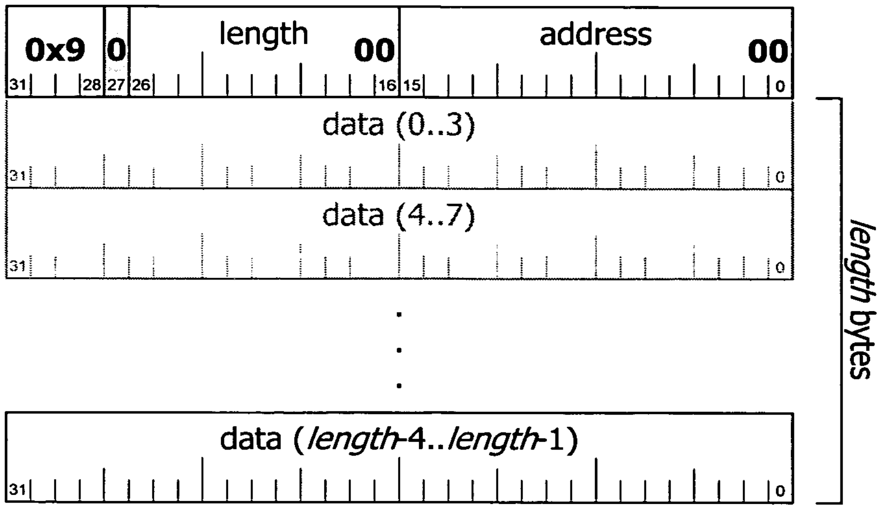
If *gen_ack* is 0, there is no explicit response from the FPGA.
If the *gen_ack* field is set to *1*, the response packet below will be generated by the FPGA and sent back to the system.

Response (from the converter)



The *length* and *address* fields echo the values from the original request packet.

PCI DMA FORWARDING (0x9)

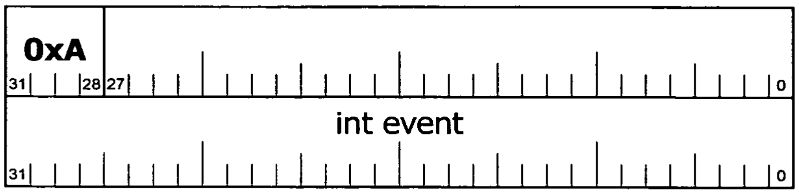


Whenever data is written to the PCI RAM or PCI-mapped FIFO memory regions in FPGA, each burst will be framed with a header as shown above, and then forwarded to the system.

Length holds the amount of *data* following the header in bytes, and is always a multiple of 4. *address* is the memory location of the first data dword.

This packet does not follow the request/response scheme, since it is not triggered by a specific action from the system.

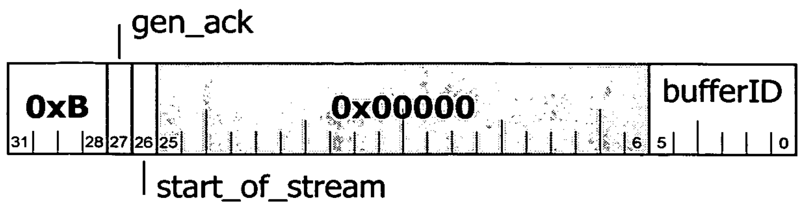
PCI INTERRUPT FORWARDING (0xA)



When the PCI INTA signal gets asserted (i.e. a falling edge is detected), the FPGA will query the Interrupt Event register (masked version) of the OHCI controller and forward the interrupt event status using the packet defined above. It will also clear the active interrupt events. It will not clear the BUSRESET interrupt event (bit 17).

WAIT FOR PCI BUFFER ACCESS (0xB)

<u>Request (from the system)</u>



This packets pauses the isochronous transmit pipe until the OHCI controller accesses (i.e. reads from or writes to) the buffer with the specified ID. The pipe is unpaused when the specified buffer ID or one of the three following buffers is accessed (in case the OHCI controller decided to skip a specific buffer, e.g. due to a bus reset).

Example: A wait for bufferID 12 will be satisfied by an access to ID 12, 13, 14 or 15.

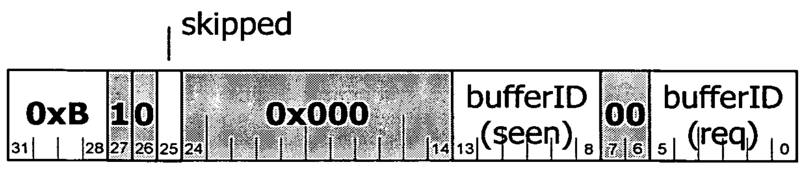
The buffer ID for a OHCI controller access is determined according to the addressing scheme outlined in section “PCI Addresses” above.

The *start_of_stream* bit shall be set for the first packet of a new stream. This will invalidate the FPGA’s internal “last buffer seen” register, which could contain a stale value from a previous streaming operation.

When the *gen_ack* bit is set, a response packet is generated (see below).

Note: The request packet consists of a single Dword, only.
Note: This packet may only be used on the isochronous transmit pipe.

<u>Response (from converter)</u>



The optional response packet echos the requested buffer ID (“req”), and actual buffer ID observed (“seen”). The *skipped* flag is set when the buffer wait request was skipped while *isoch_store_disable* was active (see the FPGA Register).